Combinational Logic

- Translates a set of Boolean inputs (0 or 1) by a mapping function (using Boolean operations) to produce a set of Boolean outputs (0 or 1).

\[ f_0 = F_0(i_0, i_1, \ldots, i_{n-1}) \]
\[ f_1 = F_1(i_0, i_1, \ldots, i_{n-1}) \]
\[ \vdots \]
\[ f_{m-1} = F_{m-1}(i_0, i_1, \ldots, i_{n-1}) \]

Design of Combinational Circuits

- Determine input, output and the relationships connecting them
- Build truth table
- Develop K-map & simplify
- Alter/simplify as necessary or desired
- Derive Boolean function and circuit diagram

CC Design Example

- \( F = 1 \) if \((ABC)_2 \geq 3\), 0 otherwise

Truth Table:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

K-map:

\[ F = A + BC \]

Diagram:
**Multiplexers**

- Multiplexer: combinational circuit with $2^n$ data inputs, $n$ control inputs that select one of the data inputs as output
- Usage example: Loading the PC (Program Counter) from binary counter or instruction register or output from ALU
- Example multiplexer: Eight-input multiplexer has 3 control inputs and 8 data inputs
  - can be packaged in a 14-pin chip: 8 data inputs, 3 control inputs, 1 output, 1 ground, 1 power

**16-bit Address PC**

**Eight-Input 1-Bit Multiplexer**

\[ F = \overline{A}\overline{B}\overline{C}D_0 + \overline{A}BCD_1 + A\overline{B}\overline{C}D_2 + \]
\[ \overline{A}B\overline{C}D_3 + A\overline{B}C\overline{D}_4 + AB\overline{C}D_5 + \]
\[ A\overline{B}C\overline{D}_6 + ABCD_7 \]

MUX can implement any Boolean function!
**MUX Implementation of F**

For example: \( F(A,B,C,D) = \sum m(1,4,5,8,10,12,13) \)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>D</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>D</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>D</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Decoders**

- Decoder: a combinational circuit that takes an n-bit number as input and uses it to select one of \( 2^n \) output lines.
- Usage example: To select one of many memory chips (see slide 12).

**3 - 8 Bit Decoder**

- \( D_0 = \overline{A}BC \)
- \( D_1 = \overline{A}BC \)
- \( D_2 = \overline{A}BC \)
- \( D_3 = \overline{A}BC \)
- \( D_4 = AB\overline{C} \)
- \( D_5 = AB\overline{C} \)
- \( D_6 = AB\overline{C} \)
- \( D_7 = AB\overline{C} \)

**1K-Byte ROM Memory**

2-4 decoder controlling references to a memory consisting of four 256 x 8-bit RAM chips:
8-Bit Shifter [1]

- Arithmetic circuit: 8-bit shifter with 8 data inputs ($D_0 - D_7$), one control input (shift left or right?, $C$) and 8 outputs.
- Right shift: $11111111$ becomes $01111111$
- Left shift: $11111111$ becomes $11111110$
- When $C$ is 1 shift right, $C$ is 0 shift left

8-Bit Shifter [2]

- Diagram of 8-bit shifter with control input $C$ and data inputs $D_0 - D_7$.
- Equation: $S_0 = D_0$, $S_1 = D_1$, $S_2 = D_2$, $S_3 = D_3$, $S_4 = D_4$, $S_5 = D_5$, $S_6 = D_6$, $S_7 = D_7$ + $A0 = 0$ & $A1 = A$ & $A + 0 = A$

Comparator Circuit

- Comparator is a combinational circuit that compares 2 input words and produces 1 if they are equal and 0 if not.
- Example circuit is a 4-bit comparator and uses 4 XOR gates and 1 NOR gate.
- Cost = 4 XOR, 1 NOR, 12 inputs = 17

Half-Adder

- Arithmetic circuit: 1-bit half adder (doesn't have a carry as input).
  - Can be used as basic block of a full-adder and chained together to create 8, 16, ... bit adders.
- Truth table:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Sum</th>
<th>Carry</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
- Equation: $Sum = A \oplus B$

Cost = 4 XOR, 1 NOR, 12 inputs = 17
**Full-Adder Design**

- Arithmetic circuit: 1-bit full-adder

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Carry in</th>
<th>Sum</th>
<th>Carry out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

SUM:

\[
\begin{array}{cccc}
00 & 01 & 11 & 10 \\
\hline
0 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
\end{array}
\]

\[
S = \overline{A}BC + \overline{A}BC + \overline{A}BC + ABC
\]

\[
C_0 = AB + AC + BC
\]

---

**Full-Adder From SOP**

- Arithmetic circuit: 1-bit full-adder from Truth Table & K-Map

Cost = 4 AND, 1 OR, 16 inputs = 21

Cost = 3 AND, 1 OR, 9 inputs = 13

---

**Full-Adder (2 Half-Adders)**

- Arithmetic circuit: 1-bit full-adder (2 half-adders with carry in)

\[
S = \overline{A}BC + \overline{A}BC + \overline{A}BC + ABC
\]

\[
S = (AB + AB)C + (AB + AB)C
\]

\[
S = (A \oplus B)C + (A \oplus B)C
\]

\[
S = (A \oplus B) \oplus C
\]

\[
C_0 = AB + AC + BC
\]

\[
C_0 = AB + A(B+B)C + (A+A)BC
\]

\[
C_0 = AB(1+C) + (AB + AB)C
\]

\[
C_0 = AB + (A \oplus B)C
\]

Cost = 2 XOR, 2 AND, 1 OR, 10 inputs = 15

---

**Multi-Bit Ripple-Carry Adder**

- Arithmetic circuit: (e.g., 4-bit adder)

---
Multi-Bit Carry-Select Adder

• Arithmetic circuits: (e.g., 32-bit adder)
  - 3 16-bit adders, lower-half and 2 upper-halves (U0, U1)
  - 0 fed into U0, 1 fed into U1. The correct upper half selected from lower half answer (one computation is wasted but it’s still faster than ripple-carry adder)
  - Replicate parallelism (have many upper halves) to get more speed (and more wasted computations)

Multi-Bit Carry-Lookahead Adder

• Arithmetic circuits:
  - Compute carry bits without reference to carry bits of previous stages
    • $C_0 = A_0B_0$ (no carry)
    • $C_1 = A_1B_1 + A_1C_0 + B_1C_0$
      $= A_1B_1 + A_1A_0B_0 + B_1A_0B_0$
    • $C_2 = A_2B_2 + A_2A_1B_1 + A_2A_1A_0B_0 + A_2B_1A_0B_0$
      $+ B_2A_1B_1 + B_2A_1A_0B_0 + B_2B_1A_0B_0$
  - Complexity and compute time increase as additional bits are added
  - Typically done in 4-8 bit units

32-bit Adder Using Four 8-Bit Adders

Simplified Arithmetic Logic Unit (ALU)

• Arithmetic circuit: ALU (Arithmetic Logic Unit)
  - Can compute $A \text{ AND } B$, $A \text{ OR } B$, $\overline{B}$, $A + B$ (add)
  - Has 2 data inputs ($A$, $B$), 2 control inputs ($F_0$, $F_1$) to select 1 of four functions above (AND, OR, NOT, addition). So, 2 bit ALU.
  - In circuit on next slide, assume INVA (inverse of A) is 0, ENA (enable input A) and ENB (enable input B) are 1.
Simplified ALU Circuit